COMSC-210 Lecture Topic 12
Recursion

Reference
Childs Ch. 13

Functions That Call Themselves
sometimes used in binary tree implementations (ch.15)

Using Data Structures
recursive loops *without* the function calls
1. done if nothing to do
2. create and seed a data structure
3. process the data structure
   popping and pushing
4. finalize the result

Recursive Factorial Function
"base case": 0! and 1! are both 1 by definition
2! is 2 times 1!
   so factorial function with 2 parameter...
   ...could call itself with 1 as parameter
n! is n times (n-1)!

and using data structures...
1. declare and initialize result to 1
   ...it's a value-returning function...
2. create an empty stack
3. while n>1, push n--
4. copy/pop the stack
   result *= top
5. return result

Recursive Fibonacci Function
"base case": f(0) is 0, f(1) is 1 by definition
f(2) = f(1) + f(0)
   so fibonacci function with 2 parameter...
   ...could call itself *twice*
and using data structures...

Recursive Binary Tree Search
1. push top node onto a "to do list" stack
2. while "to do list" is not empty
   3. copy/pop top of "to do list"
   4. if match, return "true"
   5. push left child (if there is one)
   6. push right child (if there is one)
7. return false (not found if I get this far)

```
struct Node
{
  T data;
  Node* left;
  Node* right;
};
```

"base cases": zero nodes or matching root node
   zero: `root == 0`, no match
   one: `root->data` matches (?)
if no match, search starting from `root->left`
then if no match, search starting from `root->right`

Modeling Arrival Rate In Time-step Simulations
Typically it is specified that service "requests" arrive at the average rate of "n" per time step, where "n" can be any floating point value. A "time step" models one second of "real time". But note that if, for example, the average arrival rate of service requests is 2 per time step, that does not mean that 2 arrive every time step. Sometimes none arrive in a given time step, and other times 5 may arrive -- it's just that the average over time is 2 per time step. The proper way to handle this is to use the Poisson distribution to generate randoms numbers that represent the #of requests each time step.

In the Poisson process, given an average service request rate, there is a calculable probability of zero arrivals in any given time step, another probability of one, another of two, etc. By the time you add these up towards infility, the total probability is one -- which means one of these is going to happen! In computer simulation we use the random number generator to select the #of requests in a time step. Here is a function that returns a random number of actual service requests in a period given the average service request rate:

```
// requires cmath and cstdlib (and srand and rand to seed the random number generator)
int getRandomNumberOfServiceRequests(double averageRequestsPerTimeStep)
{
  int requests = 0;
  double probOfnRequests = exp(-averageRequestsPerTimeStep);
```

```
  for (double randomValue = (double)rand() / RAND_MAX;
    (randomValue -= probOfnRequests) > 0.0;
    probOfnRequests *= averageRequestsPerTimeStep / (double)++requests);
  return requests;
}
```

NOTE: If you use `srand` in a program, make sure to call it *only once*, and call rand() once to "prime" it. The best place to put the call to srand is as the first statement in main.